Arithmetic of τ-adic Expansions for Lightweight Koblitz Curve Cryptography

Järvinen, Kimmo

2018-11

Järvinen , K , Roy , S S & Verbauwhede , I 2018 , ' Arithmetic of τ-adic Expansions for Lightweight Koblitz Curve Cryptography ' , Journal of Cryptographic Engineering , vol. 8 , no. 4 , pp. 285-300 . https://doi.org/10.1007/s13389-018-0182-0

http://hdl.handle.net/10138/308154
https://doi.org/10.1007/s13389-018-0182-0

acceptedVersion

Noname manuscript No.
(will be inserted by the editor)

Arithmetic of τ-adic Expansions for Lightweight Koblitz Curve Cryptography

Kimmo Järvinen · Sujoy Sinha Roy · Ingrid Verbauwhede

the date of receipt and acceptance should be inserted later

Abstract Koblitz curves allow very efficient elliptic curve cryptography. The reason is that one can trade expensive point doublings to cheap Frobenius endomorphisms by representing the scalar as a τ-adic expansion. Typically elliptic curve cryptosystems, such as ECDSA, also require the scalar as an integer. This results in a need for conversions between integers and the τ-adic domain, which are costly and hinder the use of Koblitz curves on very constrained devices, such as RFID tags, wireless sensors, or certain applications of the Internet-of-Things. We provide solutions to this problem by showing how complete cryptographic processes, such as ECDSA signing, can be completed in the τ-adic domain with very few resources. This allows outsourcing conversions to a more powerful party. We provide several algorithms for performing arithmetic operations in the τ-adic domain. In particular, we introduce a new representation allowing more efficient and secure computations compared to the algorithms available in the preliminary version of this work from CARDIS 2014. We also provide datapath extensions with different speed and side-channel resistance properties that require areas from less than one hundred to a few hundred gate equivalents on 0.13 μm CMOS. These extensions are applicable for all Koblitz curves.

Keywords Elliptic curve cryptography, Koblitz curves, lightweight cryptography, ECDSA

K. Järvinen is with University of Helsinki, Department of Computer Science, Gustaf Hällströmin katu 2b, 00560 Helsinki, Finland, E-mail: kimmo.u.jarvinen@helsinki.fi. S. Sinha Roy and I. Verbauwhede are with KU Leuven ESAT/COSIC and imec, Kasteelpark Arenberg 10 bus 2452, B-3001 Leuven-Heverlee, Belgium, E-mail: Firstname.Lastname@esat.kuleuven.be. This work was done when K. Järvinen was also with KU Leuven.

1 Introduction

Elliptic curve cryptography (ECC) [29,22] offers high security levels with short key lengths and relatively low amounts of computation. Hence, it is one of the most feasible alternatives for implementing public-key cryptography on constrained devices where resources (e.g., circuit area, power, and energy) are extremely limited. Such lightweight implementations of public-key cryptography are required, e.g., in wireless sensor network nodes, RFID tags, smart cards, and devices for the Internet-of-Things. For an example of an academic work on lightweight public-key cryptography tags see, e.g, [35]. One example of practical use cases of lightweight ECC are German identification documents and passports (see, e.g., [36]). Several researchers have proposed implementations which aim to minimize area, power, and/or energy of computing elliptic curve scalar multiplications [4,6,17,24,26] which are the fundamental operations of all elliptic curve cryptosystem.

Koblitz curves [23] are a special class of elliptic curves which allow very efficient elliptic curve operations when scalars used in scalar multiplications are given as τ-adic expansions. Koblitz curves allow extremely fast scalar multiplications on both software [42,14,40,3,16,13] and hardware [33,27,2,18,5,12,11]. A recent paper [4] showed that they can be implemented also with very few resources (especially, in terms of energy) if the scalars are already in the τ-adic domain. Many cryptosystems require both the integer and τ-adic representations of the scalar which results in a need for conversions between the domains. Most hardware implementations of the conversions [19,9,10,1,37] require a lot of resources making them infeasible for constrained devices. This has prevented from using Koblitz curves although they would otherwise result in very efficient

lightweight implementations. The only exception is the converter recently presented in [38]. A workaround to the problem is to design a protocol that operates directly in the τ-adic domain [8]. However, this approach prevents from using standardized algorithms and protocols which, consequently, makes the design work more laborious and may even lead to cryptographic weaknesses in the worst case.

In this paper, we show how the computationally weaker party of a cryptosystem can delegate conversions to the more powerful party by computing all operations directly in the τ-adic domain with a small datapath extension for τ-adic arithmetic. The approach is applicable to Koblitz curve cryptosystems that require scalar multiplications and modular arithmetic with the scalar, e.g., Elliptic Curve Digital Signature Algorithm (ECDSA). This can be done without affecting the cryptographic strength of the cryptosystem. To summarize, we show how Koblitz curves can be used more efficiently in lightweight implementations.

A preliminary version [20] of this paper was published in CARDIS 2014. The novel contributions of this extended version are the following:

- We provide further details and more comprehensive analysis of the approach presented in [20];
- We introduce a representation called partial τ-adic expansion which allows τ-adic arithmetic without expensive foldings and leads to faster and more secure implementations;
- We explore how the circuitries for the algorithms can be unrolled in order to obtain speedups with only small increases in area requirements;
- We propose algorithms which are protected against single-trace side-channel attacks such as timing attacks and simple power analysis; and
- We present a lightweight implementation of an existing conversion algorithm from [10]. We compare τ-adic arithmetic to this implementation and the converter from [38] and show that τ-adic arithmetic has certain advantages and offers tradeoffs which are not available with conversion based approaches.

This paper is structured as follows. Sect. 2 discusses Koblitz curves and ECDSA. Sect. 3 explores existing options to implement Koblitz curves in lightweight applications and introduces the idea of outsourcing conversions. An addition algorithm for the τ-adic domain is presented and analyzed in Sect. 4. Sect. 5 presents algorithms for other arithmetic operations. Sect. 6 introduces the partial τ-adic expansion and algorithms based on it. Sect. 7 presents datapath extensions for the algorithms from Sects. 4–6. Lightweight implementations of existing conversion algorithms are described in Sect. 8 for fair comparisons. Sect. 9 presents results on $0.13\,\mu$m CMOS and compares them to the lightweight converters. Sect. 10 closes the paper with conclusions.

2 Koblitz Curves and ECDSA

In the following, we discuss ECC and Koblitz curves and, then, present ECDSA signature generation as an example.

2.1 Elliptic Curve Cryptography and Koblitz Curves

In the mid-1980s, Miller [29] and Koblitz [22] showed how public-key cryptography can be based on the difficulty of solving discrete logarithms in an additive Abelian group $\mathcal{E}$ formed by points on an elliptic curve. Let $k \in \mathbb{Z}_+$ and $\mathbf{P} \in \mathcal{E}$. The main operation in ECC is scalar multiplication given by:

$$k\mathbf{P} = \underbrace{\mathbf{P} + \mathbf{P} + \ldots + \mathbf{P}}_{k \text{ times}} \ . \tag{1}$$

The operation $\mathbf{Q} + \mathbf{R}$, where $\mathbf{Q}, \mathbf{R} \in \mathcal{E}$, is called point addition if $\mathbf{Q} \neq \pm\mathbf{R}$ and point doubling if $\mathbf{Q} = \mathbf{R}$. Scalar multiplication can be computed with a series of point doublings and point additions, e.g., by using the well-known double-and-add algorithm. Elliptic curves over $GF(2^m)$, finite fields of characteristic two, are often preferred in hardware implementations of ECC because of the efficient carry-less arithmetic. These curves are called binary curves.

Koblitz curves [23] are the following binary curves:

$$y^2 + xy = x^3 + a_2x^2 + a_6 \tag{2}$$

where $a_2 \in \{0, 1\}$, $a_6 = 1$, and $x, y \in GF(2^m)$. Let $\mathcal{K}$ denote the Abelian group of points (x, y) that satisfy (2) together with $\mathcal{O}$, which is a special point that acts as the zero element of the group. Koblitz curves have the property that if a point $\mathbf{P} = (x, y) \in \mathcal{K}$, then also its Frobenius endomorphism $F(\mathbf{P}) = (x^2, y^2) \in \mathcal{K}$. This allows devising efficient scalar multiplication algorithms where Frobenius endomorphisms are computed instead of point doublings. It can be shown that $F(F(\mathbf{P})) - \mu F(\mathbf{P}) + 2\mathbf{P} = 0$, where $\mu = (-1)^{1-a_2}$, holds for all $\mathbf{P} \in \mathcal{K}$ [23]. Consequently, $F(\mathbf{P})$ can be seen as a multiplication by the complex number τ that satisfies $\tau^2 - \mu\tau + 2 = 0$, which gives $\tau = (\mu + \sqrt{-7})/2$.

If the scalar k is given using the base τ as a τ-adic expansion $K = \sum K_i\tau^i$, the scalar multiplication $K\mathbf{P}$ can be computed with a Frobenius-and-add algorithm, where Frobenius endomorphisms are computed for each K_i and point additions (or subtractions) are computed for $K_i \neq 0$. This is similar to the double-and-add algorithm except that computationally expensive point

doublings are replaced with cheap Frobenius endomorphisms. Hence, if a τ-adic expansion can be efficiently found, then Koblitz curves offer significant efficiency improvements compared to general binary curves.

We use the following notation. Lower-case letters $a, b, c, \ldots$ denote integers and upper-case letters $A, B, C, \ldots$ denote τ-adic expansions. If both versions of the same letter (e.g., a and A) are used in the same context, then the values are related; to state this explicitly, we denote $A \stackrel{\circ}{=} a$. Bold-faced upper-case letters $\mathbf{P}, \mathbf{Q}, \ldots$ denote points on elliptic curves.

2.2 ECDSA Signature Generation

An ECDSA signature (r, s) for a message $\mathcal{M}$ is computed as follows [32]:

$$k \in_R [1, q-1] \tag{3}$$
$$r = [k\mathbf{P}]_x \tag{4}$$
$$e = H(\mathcal{M}) \tag{5}$$
$$s = k^{-1}(e + dr) \bmod q \tag{6}$$

where q is the order of $\mathbf{P}$, d is the signer's private key, $[k\mathbf{P}]_x$ is the x-coordinate of $k\mathbf{P}$, and $H(\mathcal{M})$ is the hash of $\mathcal{M}$.

Equation (4) is efficiently computed using Koblitz curves if k is given as a τ-adic expansion; i.e., we compute $r = [K\mathbf{P}]_x$. In this paper, we assume that the coefficients of K take values $K_i \in \{-1, 0, 1\}$, e.g., K can be represented with the τ-adic nonadjacent form (τNAF) [39] or the τ-adic zero-free representation (τZFR) [34,41]. The τNAF gives improvements in computation latency and the τZFR offers protection against side-channel attacks. Both of them can be encoded with m bits by using the encoding proposed by Joye and Tymen [21] or by storing only the signs of the coefficients, respectively. In addition to the scalar multiplication, the signing requires modular integer arithmetic in (6). Hence, we need both the integer k and the τ-adic expansion K.

We can avoid the expensive inversion of (6) by transmitting the numerator and denominator separately after blinding them with $b \in_R [1, q-1]$ [31]:

$$s_n = b(e + dr) \bmod q \tag{7}$$
$$s_d = bk \bmod q \ . \tag{8}$$

We use this technique for efficiency reasons, but the proposed idea and techniques apply also without it. Although we focus on ECDSA, the proposed idea and algorithms apply also to other Koblitz curve cryptosystems, e.g., Schnorr signatures.

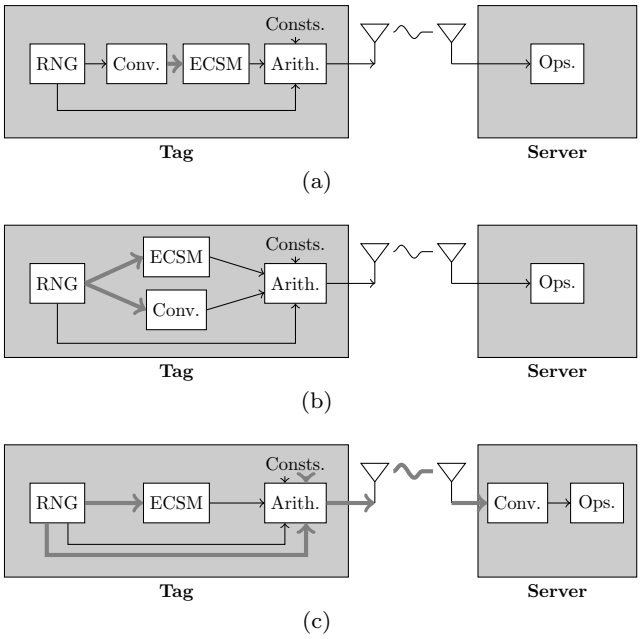

Fig. 1 Three options for using Koblitz curves in a wireless tag. Thin black and thick gray arrows represent integer and τ-adic values, respectively. (a) the random number generator (RNG) generates an integer k which is converted to a τ-adic expansion K for the elliptic curve scalar multiplication (ECSM) and k is used for the arithmetic part; (b) the RNG generates a random K for the ECSM which is converted to k for the arithmetic part; and (c) the RNG generates a random K but the arithmetic part is also performed (at least partly) in the τ-adic domain. The conversion is delegated to the more powerful server. In addition to k (or K), the RNG is used also for obtaining other random variables in the cryptosystem.

3 Koblitz Curves in Lightweight Applications

Lightweight applications are typically asymmetric in the sense that one of the communicating parties is strictly limited in resources whereas the other is not. As an example, we consider an application where a wireless tag communicates with a server over a radio channel. The tag is limited in computational resources, power, and energy but the server has plenty of resources for computations. The tag implements a Koblitz curve cryptosystem which requires both elliptic curve operations and modular arithmetic with integers.

This sections explores solutions for implementing lightweight Koblitz curve cryptosystems that require both scalar multiplications and arithmetic with the scalar. We survey two existing options for computing ECDSA signatures on Koblitz curves in Sect. 3.1 as well as the new idea for delegating conversions from the tag to the server in Sect. 3.2.

3.1 Solutions Based on Conversions

The first option, which is depicted in Fig. 1(a), is to generate k as a random integer and convert it into a τ-adic expansion K for scalar multiplication (4). Equation (6) or (8) can be computed using the original integer k. The first method for conversion was given by Koblitz [23]. It has the drawback that τ-adic expansions are twice as long as the original scalars. Later, Meier and Staffelbach [28] and Solinas [39] showed that expansions of approximately the same length as the original scalar can be found. Solinas [39] also introduced τNAF and windowed τNAF (w-τNAF) representations. These conversions require, e.g., operations with large rational numbers which render them very inefficient for hardware implementations. The first hardware oriented conversion algorithm and implementation was presented by Järvinen et al. [19]. Brumley and Järvinen [10] later presented an algorithm requiring only integer additions and it has been used as the basis of all state-of-the-art converters. However, if their algorithm is implemented in a straigthforward manner, it becomes too large for very constrained devices mostly because it uses long adders and a large number of registers. Their work was extended by Adikari et al. [1] and Sinha Roy et al. [37] who focused on improving speed at the expense of resource requirements, which makes them even less suitable for constrained devices. The first lightweight conversion algorithm and implementation were recently proposed by Sinha Roy et al. [38]. We compare our results to this work later in Sect. 9.

The second option, which is shown in Fig. 1(b), is to generate the scalar as a random τ-adic expansion K and to find its integer equivalent for computing (6) or (8). Generating random τ-adic expansions was first mentioned (and credited to Lenstra) by Koblitz [23] but he did not provide a method for finding the integer equivalent of the scalar. The first method for retrieving the integer equivalent k was proposed by Lange in [25]. Her method requires several multiplications with long operands. More efficient methods were later introduced by Brumley and Järvinen in [9,10]. We design our own lightweight implementation of the algorithm from [9, 10] in Sect. 8 and use it for comparisons in Sect. 9.

3.2 Solution for Outsourcing Conversions

A third option, which is shown in Fig. 1(c), was introduced in the preliminary version of this paper that was presented in CARDIS 2014 [20]. Similarly to the second option, the tag generates a random τ-adic expansion K and uses it for scalar multiplication (4). However, the tag does not compute the integer equivalent k but, instead, uses K directly and computes (6) or (8) in the τ-adic domain. The results of these operations (τ-adic expansions) are transmitted over the radio channel to the server which first converts the results to integers and then proceeds with normal server-side operations. The values that do not depend on the scalar, i.e. (7), should be computed with modular integer arithmetic because it is cheaper. Scalar multiplication is still computed entirely using binary field arithmetic and it does not require any modifications because of the use of τ-adic arithmetic for processing the scalar k. Clearly, this option improves efficiency of the tag only if operations in the τ-adic domain are cheaper than conversions. In the following, we show that they can, indeed, be implemented with very few resources. From security perspective, the third option is equivalent with the second option (see, e.g., [25]) because transmitting τ-adic expansions instead of their integer equivalents does not reveal any additional information about the secret scalars.

The idea has similarities with [8] where a modified version of the Girault-Poupard-Stern identification scheme was built on τ-adic expansions. Both [8] and the new idea use arithmetic in the τ-adic domain. We adapt and further develop the addition algorithm from [8]. The new idea allows delegating conversions to the more powerful party for arbitrary Koblitz curve cryptosystems requiring scalar multiplications and modular integer arithmetic with the scalar, whereas [8] presented a single identification scheme built around τ-adic expansions only. For instance, it is unclear how to build a digital signature scheme that uses only τ-adic expansions because the ideas of [8] cannot be directly generalized to other schemes. We also provide the first hardware realizations of algorithms required for τ-adic arithmetic. These implementations may have importance also for implementing the scheme from [8].

The rest of the paper focuses primarily on efficient computation of (8): $b \times K$, where b is an integer and K is a τ-adic expansion. This allows computing ECDSA signatures in low-resource tags as shown above. Also algorithms for implementing other arithmetic operations in the τ-adic domain are provided for completeness. This allows using the new idea for a variety of Koblitz curve cryptosystems.

4 Addition in the τ-adic Domain

The cornerstone of the idea discussed in Sect. 3.2 is to devise an efficient algorithm for adding two τ-adic expansions. In this section, we show how to construct such an algorithm. Our addition algorithm has similarities with the algorithm from [8] which also com-

putes additions of τ-adic expansions. Our algorithm is more efficient because it avoids unnecessary steps and uses simpler methods for deriving C_i, t_0, and t_1. We also provide a deeper analysis of the algorithm. Other arithmetic operations can be built upon the addition algorithm and they are discussed later in Sect. 5.

Let A and B be the τ-adic expansions of two positive integers a and b such that

$$A = \sum_{i=0}^{n-1} A_i \tau^i \quad \text{and} \quad B = \sum_{i=0}^{n-1} B_i \tau^i \tag{9}$$

where $A_i \in \{0,1\}$ and $B_i \in \{-1,0,1\}$ so that $A_{n-1} = 1$ and/or $B_{n-1} = \pm 1$. Signed bits are allowed for B for two reasons: (a) Koblitz curve cryptosystems are typically implemented by using representations with signed bits (e.g., τNAF or τZFR) and (b) this allows computing subtractions with the same algorithm.

Coefficient-wise addition of the two expansions gives:

$$C = A + B = \sum_{i=0}^{n-1} C_i \tau^i \tag{10}$$

where $C_i = A_i + B_i \in \{-1,0,1,2\}$. This expansion is correct in the sense that $C \stackrel{\circ}{=} a + b$ but the set of digit values has grown. Hence, the expansion must be processed in order to obtain a binary τ-adic expansion. Instead of allowing C to have signed binary values as in [8], we limit the set of digits to $C_i \in \{0,1\}$ in order to simplify computations and decrease the storage requirements for C. This does not imply restrictions for the use of the addition algorithm in our case as long as B_i are allowed to have signed binary values because we do not use the results of additions for computing scalar multiplications.

The binary τ-adic expansion C can be found analogously to normal addition of binary numbers by using a carry [8]. The main difference is that the carry is a τ-adic number t. A coefficient $C_i \in \{0,1\}$ is obtained by adding the coefficients A_i and B_i with the carry from the previous iteration and by reducing this value modulo 2; i.e., by taking the least significant bit (lsb). Every τ-adic number and, hence, also t can be represented as $t_0 + t_1\tau$ where $t_0, t_1 \in \mathbb{Z}$ [39]. Updating the carry for the next iteration requires a division by τ. As shown by Solinas [39], $t_0 + t_1\tau$ is divisible by τ if and only if t_0 is even. Subtracting C_i (equivalent with the rounding towards the nearest smaller integer after division by two) ensures this and, hence, we get:

$$((t_0 - C_i) + t_1\tau)/\tau = t_1 + \mu \left\lfloor \frac{t_0}{2} \right\rfloor - \left\lfloor \frac{t_0}{2} \right\rfloor \tau \ . \tag{11}$$

We continue the above process for all n bits and until $(t_0, t_1) \neq (0,0)$. The resulting algorithm is shown in Alg. 1.

Input: τ-adic expansions $A = \sum_{i=0}^{n-1} A_i\tau^i \stackrel{\circ}{=} a$ and $B = \sum_{i=0}^{n-1} B_i\tau^i \stackrel{\circ}{=} b$, parameter μ
Output: $C = \sum_{i=0}^{n'-1} C_i\tau^i$, where $C_i \in \{0,1\}$, such that $C \stackrel{\circ}{=} a + b$

```
(t0, t1) ← (0, 0); i ← 0
while i < n or (t0, t1) ≠ (0, 0) do
    r ← A_i + B_i + t0
    C_i ← r mod 2
    (t0, t1) ← (t1 + μ ⌊r/2⌋, − ⌊r/2⌋)
    i ← i + 1
return C
```

Algorithm 1: Addition in the τ-adic domain

Remark 1 Computing subtractions with Alg. 1 is straightforward: $A - B = A + (-B) = A + \sum_{i=0}^{n-1}(-B_i)\tau^i$. I.e., we flip the signs of B_i and compute an addition with Alg. 1.

4.1 Analysis of Alg. 1

There are certain aspects that must be analyzed before Alg. 1 is ready for efficient hardware implementation. The most crucial one is the size of the carry (t_0, t_1) because efficient hardware implementation is impossible without knowing the number of flip-flops required for it. The ending condition of Alg. 1 also implies that the latency of an addition depends on the values of the operands. This might open vulnerabilities against timing attacks. The following analysis sheds light on these aspects and provides efficient solutions for them.

In order to analyze Alg. 1, we model it as a finite state machine (FSM) so that the carry (t_0, t_1) represents the state. Alg. 1 can find unsigned binary τ-adic expansions with any $A_i, B_i \in \mathbb{Z}$ but, in this analysis and in the following propositions, we limit them so that $A_i \in \{0,1\}$ and $B_i \in \{-1,0,1\}$, as described above. The FSM is constructed starting from the state $(t_0, t_1) = (0,0)$ by analyzing all transitions with all possible inputs $A_i + B_i \in \{-1,0,1,2\}$. E.g., when $\mu = 1$, we find out that the possible next states from the initial state $(0,0)$ are $(0,0)$ with inputs 0 and 1 (the corresponding outputs are then 0 and 1), $(-1,1)$ with input -1 (output 1), and $(1,-1)$ with input 2 (output 0). Next, we analyze $(-1,1)$ or $(1,-1)$, and so on. The process is continued as long as there are states that have not been analyzed. The resulting FSM for $\mu = 1$ is depicted in Fig. 2 and it contains 21 states. We draw two major conclusions from this FSM (and the corresponding one for $\mu = -1$ which is omitted for brevity).

Proposition 1 *For both $\mu = \pm 1$, the carry (t_0, t_1) can be represented with 6 bits so that both t_0 and t_1 require 3 bits.*

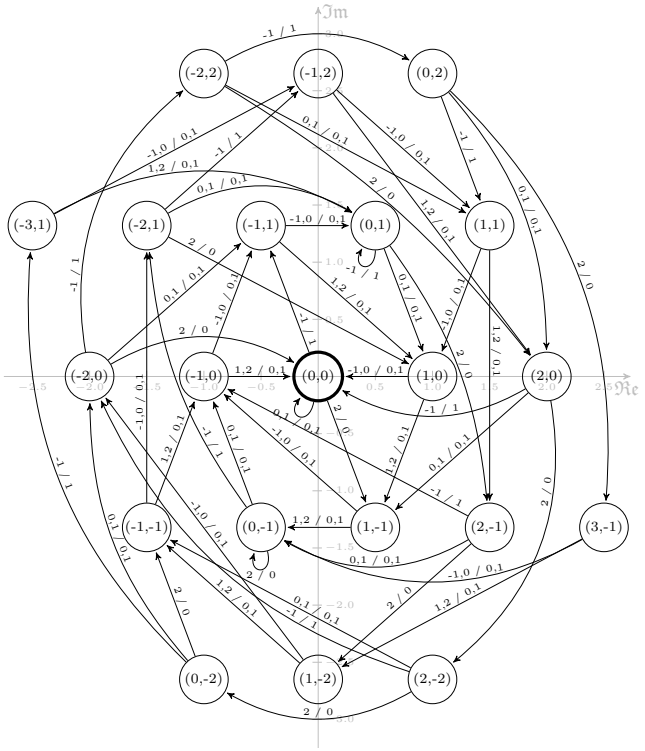

Fig. 2 The FSM for Alg. 1, when $\mu = 1$, with inputs $A_i \in \{0, 1\}$ and $B_i \in \{-1, 0, 1\}$. The FSM is plotted on the complex plane so that each state is positioned based on its complex value $t = t_0 + t_1\tau$. The states are labeled with (t_0, t_1). State transitions are marked with *in* / *out* where *in* are the inputs for the transition and *out* are the corresponding outputs.

Proof The FSM of Fig. 2 shows that $-3 \leq t_0 \leq 3$ and $-2 \leq t_1 \leq 2$. There are 7 distinct values for t_0 and 5 for t_1 and, hence, both require 3 bits. The FSM for $\mu = -1$ can be constructed similarly and it also contains 21 states so that $-3 \leq t_0 \leq 3$ and $-2 \leq t_1 \leq 2$. Hence, t_0 and t_1 both require 3 bits for $\mu = \pm 1$. Consequently, the carry requires 6 bits. □

Remark 2 The FSMs have 21 states, which can be represented with only 5 bits. Unfortunately, if we implement Alg. 1 as an FSM, the growth in the size of the combinational part outweighs the lower number of flip-flops.

Proposition 2 *Let n be the larger of the lengths of A and B; i.e., $A_{n-1} = 1$ and/or $B_{n-1} = \pm 1$. Then, Alg. 1 returns C with a length n' that satisfies*

$$n' \leq n + \lambda \tag{12}$$

where $\lambda = 7$ for both $\mu = \pm 1$.

Proof After all n bits of A and B have been processed, the FSM can be in any of the 21 states. Hence, the constant λ is given by the longest path from any state to the state $(0, 0)$ when the input is fixed to zero; i.e., $A_i = B_i = 0$. The FSM of Fig. 2 shows that the longest path starts from the state $(0, 2)$ and goes through the following states $(2, 0)$, $(1, -1)$, $(-1, 0)$, $(-1, 1)$, $(0, 1)$, and $(1, 0)$ to $(0, 0)$ and outputs $(0, 0, 1, 1, 1, 0, 1)$. Thus, $\lambda = 7$ for $\mu = 1$. It can be shown similarly that $\lambda = 7$ also for $\mu = -1$. □

5 Other τ-adic Operations

In this section, we describe algorithms for other arithmetic operations in the τ-adic domain. These algorithms use the addition algorithm given in Alg. 1.

5.1 Folding

The length of an arbitrarily long τ-adic expansion can be reduced to about m bits without changing its integer equivalent modulo q. The integer equivalent of a τ-adic expansion $A = \sum_{i=0}^{n-1} A_i \tau^i$ can be retrieved by computing the sum $a = \sum_{i=0}^{n-1} A_i s^i \pmod q$ where s, the integer equivalent of τ, is a per-curve constant integer [25]. Because $s^m \equiv 1 \pmod q$,

$$a = \sum_{i=0}^{n-1} A_i s^i \equiv \sum_{j=0}^{\lfloor n/m \rfloor} \sum_{i=0}^{m-1} A_{jm+i} s^i \pmod q, \tag{13}$$

where $A_i = 0$ for $i \geq n$. As a result of (13), an expansion can be compressed to approximately m bits by "folding" the expansion; i.e., folding is analogous to modular reduction. Let $A^{(j)} = \sum_{i=0}^{m-1} A_{jm+i} \tau^i$, the j-th m-bit block of A. Then, an approximately m-bit τ-adic expansion B having the same integer equivalent with A can be obtained by computing $B = A^{(0)} + A^{(1)} + \ldots + A^{(\lfloor n/m \rfloor)}$ with $\lfloor n/m \rfloor$ applications of Alg. 1. Because of the carry structure of Alg. 1, the length of the expansion may still exceed m bits. Additional foldings can be computed in the end in order to trim the length of B below a predefined bound $\ell \geq m$. An algorithm for folding (including the optional trimming in the end) is given in Alg. 2. Typically, the optional trimming requires at most one addition, $B^{(0)} + B^{(1)}$, often it is not needed at all.

By Proposition 2, if a folding is computed after every addition, then it becomes $A^{(0)} + A^{(1)}$ with an m-bit $A^{(0)}$ and an at most 7-bit $A^{(1)}$. While in theory this addition can give a result which is longer than m bits, the result is at most m bits long with an extremely high probability. In fact, the folding can be ended as soon as all bits of $A^{(1)}$ have been processed and $t = (0, 0)$ because, after this, all bits of the result will be the same as in $A^{(0)}$. We performed experiments on the practical lengths of folding computations. We computed $C = A + B$ where A is a random binary τ-adic expansion ($A_i \in \{0, 1\}$) and B is either

Input: τ-adic expansion $A = \sum_{i=0}^{n-1} A_i \tau^i \stackrel{\circ}{=} a$, m, and $\ell \geq m$
Output: $B = \sum_{i=0}^{n'-1} B_i \tau^i \stackrel{\circ}{=} b = a$ and $n' \leq \ell$
1 $B \leftarrow A^{(0)}$
2 **for** $j = 1$ **to** $\lfloor n/m \rfloor$ **do**
3 $\quad B \leftarrow B + A^{(j)}$ /* Alg. 1 */
4 **while** $n' > \ell$ **do**
5 $\quad B \leftarrow B^{(0)} + \ldots + B^{(\lfloor n'/m \rfloor)}$ /* Alg. 1 */
6 **return** B

Algorithm 2: Folding

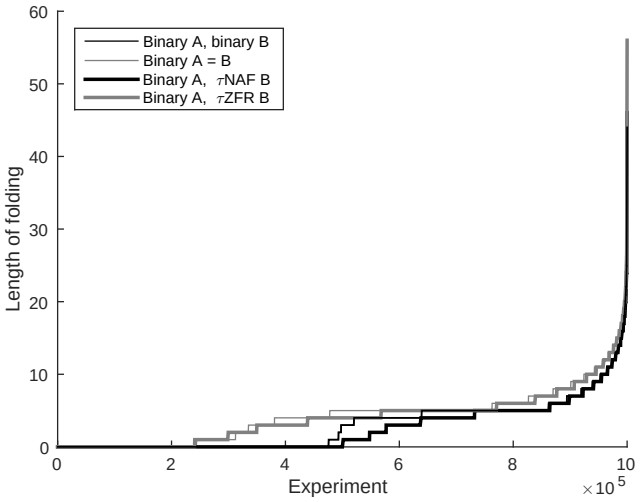

Fig. 3 Lengths of foldings after $C = A + B$ with different types of random A and B. The results of 1,000,000 experiments are sorted by the number of required iterations.

(a) a random binary τ-adic expansion ($B_i \in \{0, 1\}$),
(b) $B = A$,
(c) a random τNAF ($B_i \in \{-1, 0, 1\}$), or
(d) a random τZRF ($B_i \in \{-1, 1\}$).

The results of 1,000,000 experiments are shown in Fig. 3 so that they are ordered by the number of iterations required to complete the folding. We see that roughly 50 % of experiments did not require any folding for (a) and (c) because the addition gave an at most m-bit C. For (b) and (d), this number was about 25 %. The average numbers of iterations were only 2.96, 4.21, 2.56, and 4.00 for (a), (b), (c), and (d), respectively. Less than 10 iterations were required for 92–95 % of the experiments. The maximum number of iterations witnessed in the experiments were 44, 47, 46, and 56, respectively. The results show that the average cost of computing a folding is low if it is computed after each addition. However, if (somewhat) constant-time foldings are needed, a high number of iterations (e.g., 64 or more) needs to be computed making folding a relatively costly operation. In Sect. 6, we show that foldings can be avoided completely by using a special representation called partial τ-adic representation.

Input: τ-adic expansions $A = \tau^{n-1} + \sum_{i=0}^{n-2} A_i \tau^i \stackrel{\circ}{=} a$, where $A_i \in \{0, 1\}$, and $B \stackrel{\circ}{=} b$, where $B_i \in \{-1, 0, 1\}$
Output: $C = A \times B$ such that $C \stackrel{\circ}{=} a \times b$
1 $C \leftarrow B$ /* Alg. 1 */
2 **for** $i = n - 2$ **to** 0 **do**
3 $\quad C \leftarrow \tau C$ /* Shift */
4 $\quad$ **if** $A_i = 1$ **then**
5 $\quad\quad C \leftarrow C + B$ /* Alg. 1 */
6 **return** C

Algorithm 3: Multiplication in the τ-adic domain

Input: Integer $a = 2^{\lfloor \log_2 a \rfloor} + \sum_{i=0}^{\lfloor \log_2 a \rfloor - 1} a_i 2^i$, where $a_i \in \{0, 1\}$, and a τ-adic expansion $B \stackrel{\circ}{=} b$, where $B_i \in \{-1, 0, 1\}$
Output: C such that $C \stackrel{\circ}{=} a \times b$
1 $C \leftarrow B$ /* Alg. 1 */
2 **for** $i = \lfloor \log_2 a \rfloor - 1$ **to** 0 **do**
3 $\quad C \leftarrow C + C$ /* Alg. 1 */
4 $\quad$ **if** $a_i = 1$ **then**
5 $\quad\quad C \leftarrow C + B$ /* Alg. 1 */
6 **return** C

Algorithm 4: Multiplication by an integer in the τ-adic domain

5.2 Multiplication

Two τ-adic expansions A and B are multiplied as follows:

$$C = A \times B = \sum_{i=0}^{n-1} A_i \tau^i B \ . \tag{14}$$

An algorithm for computing (14) can be devised by using a variation of the binary method. It was also proposed in [8] that multiplications of two τ-adic expansions can be done by adopting the binary method (possibly combined with the Karatsuba approach). In Alg. 3, an addition is computed with Alg. 1 if $A_i = 1$ and a multiplication by τ is performed for all A_i by shifting the bit vector. Hence, multiplication requires $n-1$ shifts and $\rho(A)$ additions, where $\rho(A)$ is the Hamming weight of A. A bit-serial most significant bit (msb) first multiplication is presented in Alg. 3. In order to convert B into an unsigned binary τ-adic expansion, one first adds B to zero with Alg. 1 in Line 1 of Alg. 3.

The binary method can be used also for computing multiplications where the other operand, say a, is an integer. This is required, e.g., to compute $s_d = b \times K$ for ECDSA signature generation as discussed in Sect. 2. Alg. 4 presents a bit-serial msb first algorithm for computing $C = a \times B$ such that $C \stackrel{\circ}{=} a \times b$. It requires $n + \rho(A) - 1$ additions with Alg. 1.

To keep the result and intermediate values close to m bits, foldings should be computed during the algorithms. As shown in Sect. 5.1, the average cost varies,

Input: τ-adic expansion A of integer a and $q' = q - 2$
Output: B such that $b \equiv a^{-1} \pmod{q}$
1 $B \leftarrow A$ /* Alg. 1 */
2 **for** $i = \lfloor \log_2 q' \rfloor - 1$ **to** 0 **do**
3 $B \leftarrow B \times B$ /* Alg. 3 */
4 **if** $q'_i = 1$ **then**
5 $B \leftarrow B \times A$ /* Alg. 3 */
6 **return** B

Algorithm 5: Inversion modulo q in the τ-adic domain

depending on the types of the operands, from 2.56 to 4.21 iterations per addition if each addition is followed by a folding.

Remark 3 Alg. 4 also serves as an algorithm for converting integers to the τ-adic domain. An integer a can be converted by computing $a \times 1$ with Alg. 4. The algorithm returns $C = A$, the unsigned binary τ-adic expansion of a.

Remark 4 Different versions of the binary method (e.g., NAF or window) can be straightforwardly used for multiplications of τ-adic expansions (also when the other operand is an integer). Especially, using Montgomery's ladder [30] provides a constant sequence of operations (shifts and additions), which improves resistance against side-channel analysis. The scalar k is typically a nonce and the adversary is limited to a single side-channel trace. Thus, constant sequence of operations offers sufficient protection against most attacks. These issues are further explored in Sects. 6 and 7.3.

5.3 Multiplicative Inverse

The multiplicative inverse modulo q, a^{-1}, for an integer a can be found via Fermat's Little Theorem:

$$a^{-1} = a^{q-2} \pmod{q} \ . \tag{15}$$

This exponentiation gives a straightforward way to compute inversions also with τ-adic expansions. Let $q' = q - 2$. Given a τ-adic expansion A, a τ-adic expansion A^{-1} such that $A \times A^{-1} \stackrel{\circ}{=} a \times a^{-1} \equiv 1 \pmod{q}$ can be found by computing:

$$A^{-1} = A^{q'} = \prod_{i=0}^{\lfloor \log_2 q' \rfloor} A^{q'_i 2^i} \ . \tag{16}$$

Alg. 5 computes (16) by using Alg. 3.

Input: Partial τ-adic expansions (A, α) and (B, β) for integers a and b, parameter μ
Output: (C, γ), where $C = \sum_{i=0}^{m-1} C_i \tau^i$ with $C_i \in \{0, 1\}$ and $\gamma = (\gamma_0, \gamma_1)$, such that $C + \gamma_0 + \gamma_1 \tau \stackrel{\circ}{=} a + b$
1 $(t_0, t_1) \leftarrow (\alpha_0 + \beta_0, \alpha_1 + \beta_1)$
2 **for** $i = 0$ **to** $m - 1$ **do**
3 $r \leftarrow A_i + B_i + t_0$
4 $C_i \leftarrow r \bmod 2$
5 $(t_0, t_1) \leftarrow (t_1 + \mu\lfloor r \rfloor / 2, -\lfloor r \rfloor / 2)$
6 **return** $(C, (t_0, t_1))$

Algorithm 6: Addition of partial τ-adic expansions

6 Partial τ-adic Expansions

Definition 1 (Partial τ-adic expansion) A partial τ-adic expansion of a positive integer a is the tuple (A, α), where the expansion part is $A = \sum_{i=0}^{m-1} A_i \tau^i$ and the remainder part is $\alpha = (\alpha_0, \alpha_1)$ such that $\alpha_0, \alpha_1 \in \mathbb{Z}$ and $A + \alpha_0 + \alpha_1 \tau \stackrel{\circ}{=} a$.

Partial τ-adic expansions are powerful because they allow computations without foldings. To achieve this, we devise a version of Alg. 1 that takes and returns partial τ-adic expansions instead of τ-adic expansions. The difference between regular τ-adic additions with Alg. 1 and the additions of partial τ-adic expansions is highlighted by denoting the latter by $\boxplus$.

Alg. 6 gives an algorithm for computing a partial τ-adic expansion (C, γ) with binary C_i when given two partial τ-adic expansions (A, α) and (B, β). Instead of initializing the algorithm with $(0, 0)$ as in Alg. 1, we now initialize it with $\alpha + \beta$ in order to take the remainder parts into account. After this, the expansion part is computed similarly as in Alg. 1. Indeed, if one runs the algorithm until $t = (0, 0)$, then one obtains $C \stackrel{\circ}{=} a + b$. However, we run the algorithm only for m iterations and obtain (C, γ), where C is exactly m bits long and γ represents "the tail" which could be up to seven bits long (see Proposition 2). The carry (t_0, t_1) can be directly used as γ because $(t_0 + t_1\tau)\tau^m \equiv t_0 + t_1\tau \pmod{q}$ after iteration $i = m - 1$.

Because C is always m bits long, an arbitrary number of additions can be computed without foldings. However, we do not yet know if the remainder part is reasonably bounded. The following proposition sheds light on this issue. Let $\mathcal{S}_0$ denote the 21 states that can be reached in Alg. 1; i.e., the states depicted in Fig. 2 for $\mu = 1$.

Proposition 3 *Let (A, α) and (B, β) be two partial τ-adic expansions such that $A_i \in \{0, 1\}$, $B_i \in \{0, \pm 1\}$, and $\alpha, \beta \in \mathcal{S}_0$. Then, $(C, \gamma) = (A, \alpha) \boxplus (B, \beta)$ with $\gamma \in \mathcal{S}_0$ if $m > 6$.*

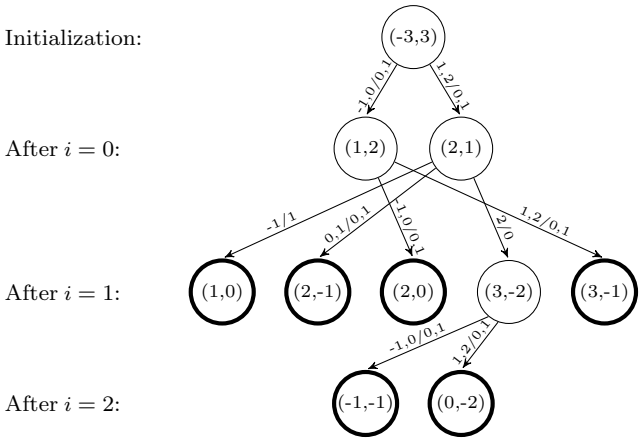

Fig. 4 The paths that Alg. 6 can take before reaching a state in $\mathcal{S}_0$ when $\alpha + \beta = (-3, 3)$ and $\mu = 1$. The states that are in $\mathcal{S}_0$ are bolded.

Proof In Line 1 of Alg. 6, t is initialized with $(\alpha_0 + \beta_0, \alpha_1 + \beta_1)$ which can yield $t \notin \mathcal{S}_0$. There are in total 69 possible states for t after Line 1 for both $\mu = \pm 1$. We denote these states by $\mathcal{S}_1$. We analyze all states $\mathcal{S}_1 \setminus \mathcal{S}_0$ separately. We compute all next states with all possible inputs $A_i + B_i \in \{-1, 0, 1, 2\}$ for as many iterations as is required until t can contain only states in $\mathcal{S}_0$. Fig. 4 shows an example of how this analysis proceeds for $t = (-3, 3)$ when $\mu = 1$. Depending on A_i and B_i, t can have two values after iteration $i = 0$: $(1, 2)$ or $(2, 1)$, neither of which is in $\mathcal{S}_0$. After iteration $i = 1$, the algorithm is in one of five possible states, of which only $(3, -2) \notin \mathcal{S}_0$. This state results in either $(-1, -1)$ or $(0, -2)$, both states in $\mathcal{S}_0$, after the next iteration $(i = 2)$. Hence, Alg. 6 is guaranteed to be in $\mathcal{S}_0$ after three iterations if it is initialized with $\alpha + \beta = (-3, 3)$. Performing similar analysis for all $\mathcal{S}_1 \setminus \mathcal{S}_0$ shows that, with all possible initializations from $\mathcal{S}_1$, it takes at most seven iterations (after $i = 6$) before Alg. 6 is in a state in $\mathcal{S}_0$. Because Alg. 6 runs for exactly m iterations, it is guaranteed to return $\gamma \in \mathcal{S}_0$ if $m > 6$. □

To summarize, Alg. 6 was shown to return bounded remainder parts for all practically relevant m. Hence, Alg. 6 can compute an arbitrary number of additions without expanding either the expansion or the remainder part. Consequently, Alg. 6 can be used for producing variants of the τ-adic arithmetic operations.

Alg. 6 specifies that A and B are exactly m bits long. If the actual length n of an expansion, say A, is smaller than m, then it can be extended to m by padding zeros: $A_i = 0$ for $n \leq i < m$. If $\alpha = (0, 0)$, then Alg. 6 can be used even for A with length up to $m + 2$. Because $A_m \tau^m \equiv A_m \pmod{q}$ and $A_{m+1} \tau^{m+1} \equiv A_{m+1} \tau \pmod{q}$, the remainder part α can store the two highest

Input: Integer $a = 2^{\lfloor \log_2 a \rfloor} + \sum_{i=0}^{\lfloor \log_2 a \rfloor - 1} a_i 2^i$, where $a_i \in \{0, 1\}$, and a τ-adic expansion $B = \sum_{i=0}^{n-1} B_i \tau^i$ with $n \leq m + 2$ such that $B \stackrel{\circ}{=} b$
Output: $C = \sum_{i=0}^{m-1} C_i \tau^i$ such that $C \stackrel{\circ}{=} a \times b$

```
1 (B, β) ← (Σ_{i=0}^{m-1} B_i τ^i, (B_m, B_{m+1}))        /* Alg. 6 */
2 (C, γ) ← (B, β)
3 for i = ⌊log_2 a⌋ − 1 to 0 do
4 |   (C, γ) ← (C, γ) ⊞ (C, γ)                           /* Alg. 6 */
5 |   if a_i = 1 then
6 |   |   (C, γ) ← (C, γ) ⊞ (B, β)                       /* Alg. 6 */
7 (C, γ) ← (C, γ) ⊞ (0, (0, 0))                          /* Alg. 6 */
8 return C
```

Algorithm 7: Multiplication by an integer by using partial τ-adic expansions

coefficients of A without changing the integer equivalent of A; i.e., one sets $(A, \alpha) = (\sum_{i=0}^{m-1} A_i \tau^i, (A_m, A_{m+1}))$.

Alg. 7 shows a variation of Alg. 4 for partial τ-adic expansions. The algorithm allows B with $B_i \in \{0, \pm 1\}$ which are $m + 2$ bits long by using the trick explained above and, hence, it can be used for τNAF expansions with length $m + a$ with $a \in \{0, 1\}$ that are commonly used in Koblitz curve cryptosystems. Line 2 of Alg. 7 stores B into the accumulator C after which the multiplication is carried out via the double-and-add approach of Alg. 4. In Line 7, a zero is added to C in order to embed the potentially nonzero γ and to obtain the m-bit binary τ-adic expansion. The probability that this addition outputs $\gamma \neq (0, 0)$ is negligible. If this nonetheless happens, then Line 7 can be repeated.

Alg. 8 presents a variation of multiplication by integer that uses Montgomery's ladder with a constant sequence of operations. The beginning and ending of the algorithm are similar to Alg. 7. The main loop computes an addition of the two accumulator values (C, γ) and (D, δ) followed by an addition where one of the accumulators is added to itself. Hence, regardless of the value of a_i two similar additions are computed on each iteration and the algorithm offers good protection against side-channel attacks.

The execution time of Alg. 8 depends on $\lfloor \log_2 a \rfloor$. This can be avoided, e.g., by adding multiples of q to a so that the sum a' has a fixed length. Alg. 8 using a' executes in constant time and returns a correct result because $a' \times B \equiv a \times B \pmod{q}$. To compute (8) for ECDSA, Alg. 8 runs in constant time simply by fixing the msb of the random b in (8) to one.

Comparing Algs. 7 and 8 reveals that there is a price to pay for constant time and operation sequence. First, Alg. 7 requires $\lfloor \log_2 a \rfloor + \rho(a) + 1 \approx \frac{3}{2} \lfloor \log_2 a \rfloor$ applications of Alg. 6 whereas Alg. 8 requires exactly $2 \lfloor \log_2 a \rfloor + 3$, which is a roughly 33 % increase in the number of additions. Second, Alg. 8 requires two accu-

Input: Integer $a = 2^{\lfloor \log_2 a \rfloor} + \sum_{i=0}^{\lfloor \log_2 a \rfloor - 1} a_i 2^i$, where $a_i \in \{0,1\}$, and a τ-adic expansion $B = \sum_{i=0}^{n-1} B_i \tau^i$ with $n \leq m+2$ such that $B \stackrel{\circ}{=} b$
Output: $C = \sum_{i=0}^{m-1} C_i \tau^i$ such that $C \stackrel{\circ}{=} a \times b$

1 $(C,\gamma) \leftarrow (\sum_{i=0}^{m-1} B_i \tau^i, (B_m, B_{m+1}))$ /* Alg. 6 */
2 $(D,\delta) \leftarrow (C,\gamma) \boxplus (C,\gamma)$ /* Alg. 6 */
3 **for** $i = \lfloor \log_2 a \rfloor - 1$ **to** 0 **do**
4 **if** $a_i = 0$ **then**
5 $(D,\delta) \leftarrow (D,\gamma) \boxplus (C,\delta)$ /* Alg. 6 */
6 $(C,\gamma) \leftarrow (C,\gamma) \boxplus (C,\gamma)$ /* Alg. 6 */
7 **else**
8 $(C,\gamma) \leftarrow (C,\gamma) \boxplus (D,\delta)$ /* Alg. 6 */
9 $(D,\delta) \leftarrow (D,\delta) \boxplus (D,\delta)$ /* Alg. 6 */
10 $(C,\gamma) \leftarrow (C,\gamma) \boxplus (0,(0,0))$ /* Alg. 6 */
11 **return** C

Algorithm 8: Montgomery's ladder for multiplication by an integer in the τ-adic domain

mulators, (C,γ) and (D,δ), whereas Alg. 7 uses only one accumulator (C,γ).

A multiplication by τ is a simple shift in Alg. 3. Multiplying a partial τ-adic expansion (A,α) by τ is more complicated because $\tau\alpha = \alpha_0\tau + \alpha\tau^2$ which cannot be used as the remainder part of the result. There are several ways to perform this multiplication. For instance, one first embeds α by computing $(B,\beta) = (A,\alpha) \boxplus (0,(0,0))$ which results in $\beta = (0,0)$ with an extremely high probability for all m of practical significance. Then, $\tau(A,\alpha) = (B_{m-1} + \sum_{i=0}^{m-2} B_i \tau^{i+1}, (0,0))$, which is a cyclic shift by one because $B_{m-1}\tau^m \equiv B_{m-1}$. Multiplications by τ^e are cyclic shifts by e because the remainder part is guaranteed to remain zero.

Multiplication of two partial τ-adic expansions can be computed by first embedding the remainder part of (A,α) by computing $(A,\alpha) \boxplus (0,(0,0))$ and, then, using Alg. 3 where shifts are replaced by the above procedure and $C + B$ with $(C,\gamma) \boxplus (B,\beta)$. If a protocol requires scalar multiplications with scalars that are given as partial τ-adic expansions, then the remainder part α should be embedded in order to avoid the problems of dealing with remainder parts in scalar multiplications. I.e., instead of computing $(K,\kappa)\mathbf{P}$, we first compute $(K',\kappa') \leftarrow (K,\kappa) \boxplus (0,(0,0))$ so that $\kappa' = (0,0)$ and then compute $K'\mathbf{P}$ in a normal way.

7 Architecture

The objective of this work was to provide a small circuitry that could be used as a datapath extension in an arithmetic logic unit (ALU) to compute τ-adic arithmetic in lightweight implementations. Fig. 5 presents datapath extensions for computing Algs. 1 and 6 for $\mu = 1$. Because $B_i \in \{-1,0,1\}$, they can be used for K with signed-bit representations (e.g., τNAF or τZFR). The datapath extensions are designed to be added into the datapath of an ALU that supports other operations required by the cryptosystem (arithmetic in $GF(2^m)$, arithmetic modulo q, etc.).

The architectures of Fig. 5 consist of registers for storing the carry (t_0,t_1) and adders for computing Lines 3-5 of Algs. 1 and 6. Proposition 1 tells that both t_0 and t_1 require three bits in Alg. 1 and, hence, Fig. 5(a) contains six flip-flops. Because $(t_0,t_1) = \alpha + \beta$ gives $-6 \leq t_0 \leq 6$ and $-4 \leq t_1 \leq 4$, they both require 4-bit registers; hence, Fig. 5(b) contains eight flip-flops. These registers must be such that they can be initialized to a specific value in order to write the values $\gamma_0 = \alpha_0 + \beta_0$ and $\gamma_1 = \alpha_1 + \beta_1$ in them. Adders for these additions are not included in the datapath extension because it is assumed that they are available in the ALU in order to compute modular arithmetic (e.g., (7)). In Fig. 5(a), r is a 4-bit value because $-3 \leq t_0 \leq 3$ and $-1 \leq A_i + B_i \leq 2$. A 5-bit r is computed in Fig. 5(b) because $-6 \leq t_0 \leq 6$. For the same reason, additional adders are required also for updating (t_0,t_1). The adders on the bottom-left compute $t_1 + \lfloor r/2 \rfloor$ and the adders on the bottom-right compute the negation: $-\lfloor r/2 \rfloor$.

Datapath extensions for $\mu = -1$ can be devised similarly but we omit the description for brevity. We merely state that they are similar to the ones for $\mu = 1$: the only difference is that the adders updating t_0 (bottom-left in Fig. 5) use the outputs of the negation circuitry that computes $-\lfloor r/2 \rfloor$ (bottom-right in Fig. 5) instead of taking $\lfloor r/2 \rfloor$ directly. Hence, the area requirements should, in theory, remain the same but the critical path becomes longer.

7.1 Unrolled Architectures

When ω iterations of the for-loops of Algs. 1 and 6 are unrolled, the logic for computing the values is replicated ω times but only a single set of registers for storing the carry (t_0,t_1) is needed in the end. Because these registers consume a significant portion of the area, unrolling gives major improvements in latency-area ratio.

Unrolling Alg. 1 is straightforward. One simply replicates the logic ω times which reduces the number of iterations to $\lceil (m+\lambda)/\omega \rceil$ (as per Proposition 2). Even if $m+\lambda$ is not a multiple of the unrolling factor ω, it suffices to pad the inputs with zeros to make the number of iterations a multiple of ω. I.e., one finds the smallest λ' such that $\lambda' \geq \lambda = 7$ and $\omega \mid m + \lambda'$.

Unrolling Alg. 6 is more complicated because it must run for exactly m iterations. Because m is a prime, $\omega \nmid m$ (with the exception of $\omega = m$) and the unrolled

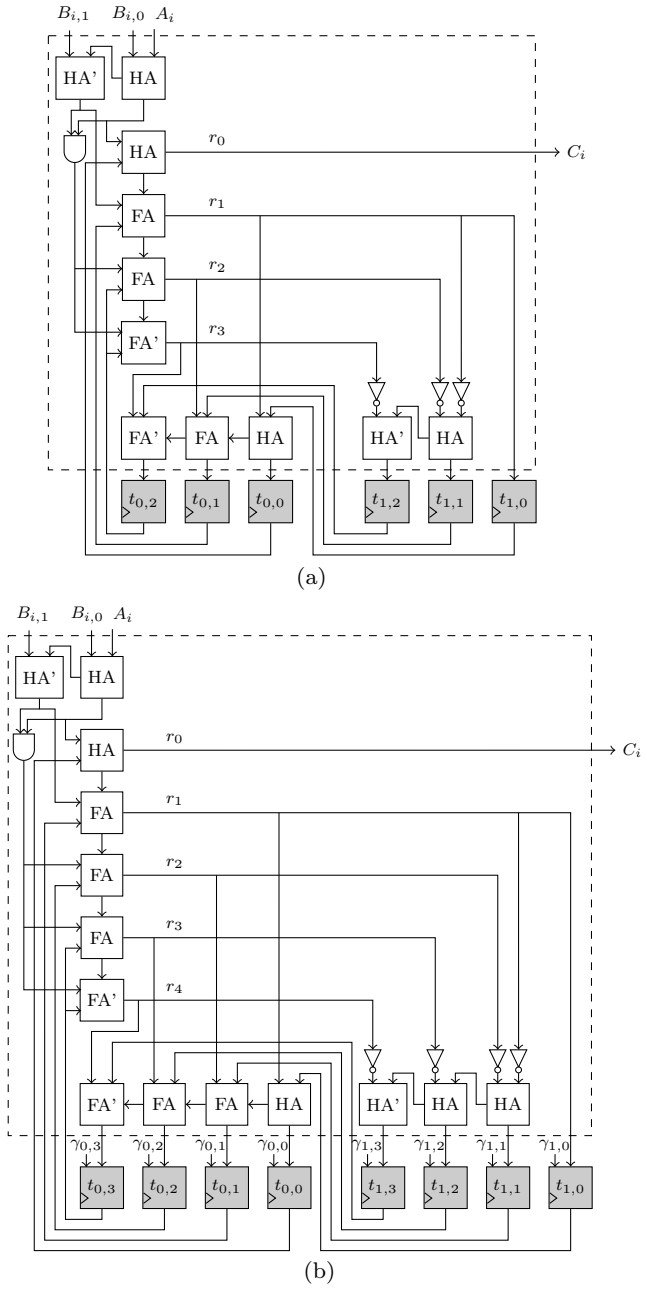

Fig. 5 Datapath extensions for (a) Alg. 1 and (b) Alg. 6 for $\mu = 1$. The circuits consist of half adders (HA), full adders (FA), half adders and full adders without carry logic (HA' and FA'), NOT and AND gates, and flip-flops. The flip-flops for (b) can be set to a specific value. All wires are single bit wires. The combinational parts that are replicated ω times in unrolled architectures are inside the dashed rectangles.

architecture must include a multiplexer for selecting values to be stored into the registers. The outputs of the unrolled iteration $m \bmod \omega$ are selected for the last iteration of the unrolled algorithm in order to store the results of the iteration m. The outputs of the last unrolled iteration are used for all other iterations.

It is possible to simplify the unrolled iterations when Fig. 5(b) is unrolled. Proposition 3 tells that if $\omega \geq 6$, then the circuitry of Fig. 5(a) can be used instead of Fig. 5(b) for the last replications of combinational parts. Already earlier replications can be optimized because the sets of possible values of t_0 and t_1 get smaller.

7.2 High-level Architecture and Latencies

In most practical cases, the datapath extension would be added to a W-bit ALU connected to a RAM which stores W-bit words. In addition to the datapath extension, Algs. 1 and 6 require also three W-bit shift registers, two for the operands A and B and one for the result C.

We consider both single-port and dual-port RAMs. In the case of a single-port RAM, reading and storing words of the operands to the shift registers requires two clock cycles. For a dual-port RAM, this can be done in a single clock cycle. An unrolled datapath extension computes ω bits of the result in one clock cycle. A natural upper bound for ω is W and to facilitate efficient implementation one should ensure $\omega \mid W$. In that case, two W-bit words are added in W/ω clock cycles. The result word is written into the RAM in one clock cycle regardless of the type of the RAM. Hence, computing one word of an addition takes $W/\omega + h$ clock cycles, where $h = 2$ or $h = 3$ for single-port and dual-port RAM, respectively. We assume that computing and storing $(t_0, t_1) = (\alpha_0 + \beta_0, \alpha_1 + \beta_1)$ and writing (t_0, t_1) to the RAM take h and one clock cycles, respectively.

Assuming $n = m$, the above procedure executes Alg. 1 in $\lceil (m+\lambda'')/W \rceil (W/\omega+h)$ clock cycles where λ'' is the smallest integer such that $\lambda'' \geq \lambda$ and $W \mid m+\lambda''$. For instance, for NIST K-163, $W = 8$, $\omega = 4$, and dual-port RAM, this gives 84 clock cycles. With practical sizes of W, a folding takes on average only $W/\omega + h$ clock cycles, which gives 4 clock cycles with the above parameters. However, constant time folding (e.g., 64 iterations) needs $(64/W)(W/\omega + h)$ clock cycles, which gives 32 clock cycles with the above parameters. Alg. 6 takes $\lceil m/W \rceil (W/\omega+h)+h+1$ clock cycles. This gives 87 clock cycles with the above parameters. Hence, Alg. 6 is constant time and roughly as fast as Alg. 1 with the non-constant time folding.

Table 1 shows the latencies of computing Algs. 7 and 8 with different unrolling factors and types of RAM for three curves from [32]: NIST K-163, K-233, and K-283.

Table 1 Latencies (clock cycles) of $b \times K$ for NIST K-163 / K-233 / K-283

		Double-and-add (Alg. 7)		Montgomery (Alg. 8)	
		Single-port RAM	Dual-port RAM	Single-port RAM	Dual-port RAM
$W = 8$	$\omega = 1$	62462 / 126332 / 184847	57072 / 115482 / 169122	83290 / 168452 / 246475	76096 / 153975 / 225496
	$\omega = 2$	42617 / 85732 / 124922	37227 / 74882 / 109197	56803 / 114280 / 166528	49609 / 99803 / 145549
	$\omega = 4$	32572 / 65432 / 94747	27182 / 54582 / 79022	43396 / 87194 / 126271	36202 / 72717 / 105292
	$\omega = 8$	27672 / 55282 / 79872	22282 / 44432 / 64147	36856 / 73651 / 106426	29662 / 59174 / 85447
$W = 16$	$\omega = 1$	52640 / 105286 / 154193	49700 / 99686 / 146118	70188 / 140386 / 205597	66264 / 132914 / 194824
	$\omega = 2$	32795 / 64686 / 94268	29855 / 59086 / 86193	43701 / 86214 / 125650	39777 / 78742 / 114877
	$\omega = 4$	22750 / 44386 / 64093	19810 / 38786 / 56018	30294 / 59128 / 85393	26370 / 51656 / 74620
	$\omega = 8$	17850 / 34236 / 49218	14910 / 28636 / 41143	23754 / 45585 / 65548	19830 / 38113 / 54775
	$\omega = 16$	15400 / 28986 / 41568	12460 / 23386 / 33493	20484 / 38580 / 55342	16560 / 31108 / 44569

7.3 Side-channel Attacks

Because the circuitry for τ-adic arithmetic processes secret values (at least K), it must be protected against side-channel attacks. We focus on side-channel properties of computing $b \times K$ for ECDSA signature. Both b and K are secret values and the cryptosystem is broken if an adversary learns either of them. Both values are also nonces meaning that they take new values for every signature generation. Hence, protection is required only against single-trace attacks (e.g., simple power analysis). In the following, we provide an algorithm level study on the side-channel properties of the proposed algorithms.

Alg. 4 scans the bits of b and utilizes the double-and-add scheme, which has a sequence of operations that depends on b ($C+C$ is computed for all b_i and $C+B$ is computed if $b_i = 1$). If $C+C$ and $C+B$ are computed as atomic operations which are indistinguishable to an adversary, then the adversary learns only the Hamming weight of b. However, if the adversary is able to distinguish these operations, then b is leaked. Alg. 1 can be considered atomic because it always runs for $m+\lambda$ similar iterations. However, foldings are required to trim the length of C to m in the course of Alg. 4. As discussed in Sect. 5.1, the simplest option is to compute a folding after each addition. This folding can be computed either for a fixed number of iterations (e.g., m or 64) or for only as many iterations that are required. The former comes with a significant performance penalty and the latter results in non-constant execution times. For the latter, Fig. 3 reveals that the length of the folding after $C + B$ differs from the folding after $C + C$. This leakage can be enough to learn information on b. Even for Montgomery's ladder, the lengths of foldings may give information about the values of the operands and leak security critical information. Hence, the algorithms that use τ-adic expansions and foldings are potentially insecure against side-channel adversaries or, alternatively, slow if one uses constant time foldings.

Partial τ-adic expansions offer constant time additions because foldings are not required. Even Alg. 7 offers some security because $C \boxplus C$ and $C \boxplus B$ are atomic operations. Hence, only the Hamming weight of b leaks through the timing side-channel. Difficulties may still arise from control logic implementing Alg. 7 (see Sect. 7.2). When the W-bit words of b are scanned by the control logic, the pattern of reading the words from the memory may reveal their Hamming weights. The adversary can learn the value of b from this information. This leakage can be avoided by using dummy reads from the memory after each bit of b.

Alg. 8 provides a constant sequence of atomic operations and offers high protection against single-trace side-channel and safe error fault attacks because it prevents the attacker from learning b and K from the pattern of operations and does not involve dummy operations. Certain recent single-trace attacks (such as horizontal collision correlation attacks [7,15]) break scalar multiplications with constant patterns of operations. In principle, Alg. 8 can be vulnerable against such attacks. However, mounting these single-trace attacks successfully against Alg. 8 can be expected to be significantly more difficult than attacking scalar multiplications because the source of leakage is much smaller ($\lceil (m/\omega) \rceil$ additions of ω-bit τ-adic expansions instead of $\lceil m/W \rceil^2$ multiplications of W-bit integers). Nevertheless, these attacks and countermeasures against them deserve further research in the future.

Because Algs. 7 and 8 are both faster and more secure than Alg. 4 (and its variant using Montgomery's ladder), we focus mainly on them in Sect. 9.

8 Discussion on Lightweight Conversions

Most of the existing hardware converters are targeted for high-speed applications and implemented on FP-

GAs. This makes fair comparisons with them very difficult. However, to show the areas of these converters and to highlight their unsuitability for lightweight applications, we have collected certain FPGA-based converters in Table 2. This table presents the smallest converters available in these publications. To put these numbers into perspective with the results later given in Table 3, one should remember, e.g., that one register is about 5.5 GE and one slice includes four registers.

The only comparable converter is the recently proposed converter [38] from the authors of this paper. It finds the τZFR for an integer k by using a datapath extension for a 16-bit ALU. It was designed specifically for NIST K-283 curve and it computes a conversion in 78,000 clock cycles. It is protected from side-channel attacks similarly to our new algorithms.

High-speed converters [10] hint that the conversion to the other direction, from a τ-adic expansion to an integer, could result in a more compact converter. In Sect. 8.1, we provide the first lightweight converter using this conversion. We use similar design decisions with the converter of [38] and the datapath extensions presented in Sect. 7 to allow fair comparisons.

8.1 τ-adic Expansion to Integer

We propose a lightweight architecture for τ-adic expansion to integer equivalent conversion based on the algorithm from [10, Fig. 6]. The algorithm computes an integer equivalent a of a τ-adic expansion A in two phases: first, repeated multiplications by τ are performed to compute an element $d_0 + d_1\tau \in \mathbb{Z}[\tau]$, then in the end, a modular multiplication $d_0 + d_1 \cdot s \mod q$ is performed to compute an integer equivalent a. See [10] for details about the algorithm.

The conversion algorithm is very simple as both multiplications by τ and s can be implemented using shifts and additions or subtractions. However, the architecture in [10] performs full-precision arithmetic to achieve high speed and hence requires a large area. To achieve very small area, we modify the algorithm to process the operands in a word-serial manner. Beside this, we also optimize the computation steps to reduce the number of additions and subtractions from three to two. In the original algorithm ([10, Fig. 6]), computation of $(d_0, d_1) \leftarrow (-2d_1 + A_i, d_0 - d_1)$ requires one subtraction from zero during the computation of $-2d_1$. We skip this subtraction by computing $(d_0, d_1) \leftarrow (2d_1 \pm A_i, d_1 - d_0)$ in the for-loop. The optimized algorithm is in Alg. 9.

In Fig. 6 we describe an architecture for computing an integer equivalent of a τZFR [34,41]. To make the design compatible with [38] and 16-bit microcontrollers,

Input: Length n, τ-adic expansion A, parameters q and s
Output: Integer equivalent a of A modulo q
1 $(d_0, d_1) \leftarrow (0, 0)$
2 **for** $i = n - 1$ **to** 0 **do**
3 $\quad d_0 \leftarrow 2d_1 + (-1)^i \cdot A_i$
4 $\quad d_1 \leftarrow d_1 - d_0$
5 $a \leftarrow (-1)^n \cdot (d_0 + d_1 \cdot s) \mod q$
6 **return** a

Algorithm 9: Computation of integer equivalent from a τ-adic expansion with $\mu = -1$

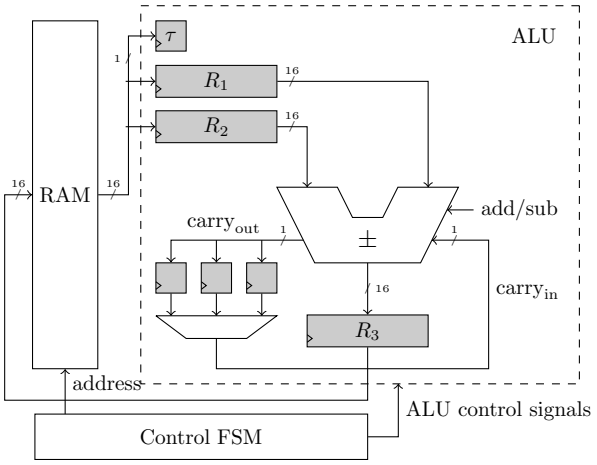

Fig. 6 Hardware Architecture for τ-adic to Integer Conversion

the datapath of the architecture is designed to process 16-bit words. The operands d_0, d_1, A, s and q are kept in the RAM. The ALU of the architecture consists of mainly adder/subtractor circuit, three 16-bit registers R_1, R_2 and R_3, three registers for storing three carry bits, and one register for storing a bit of the τ-adic expansion. Hence, a typical ALU needs to be extended with a few registers and a multiplexer.

During any iteration of the for-loop in Alg. 9, a word of the τZFR is fetched from the RAM and then stored in both R_1 and R_2. The msb of the word is the τ-bit to be processed and it is stored in the register τ-bit. Next, the word of the τZFR is left-shifted by adding the two registers R_1 and R_2 and then the result is stored in the RAM. The for-loop of Alg. 9 processes the words of d_0 and d_1 in a serial manner. In the end of the for-loop, the modular multiplication (Line 5) is performed by scanning the bits of s from left to right and performing shifts and adds depending on the bits of s. The control FSM generates address and write enable signals for the RAM and control signals for the ALU.

Alg. 9 is not protected against side-channel attacks. It involves conditional additions or subtractions depending on the loop index and the τ-adic bit. In Line 3, a subtraction of one from $2d_1$ results in a borrow prop-

Table 2 FPGA-based converters

Ref.	Description	FPGA	Latency	Area
[19]	NIST K-163, integer to τNAF converter	Altera Stratix II EP2S60F1020C4	491	1433 ALUTs + 988 Regs
[9]	NIST K-163, τ-adic to integer converter	Altera Stratix II EP2S60F1020C4	489	1057 ALUTs + 654 Regs
[10]	NIST K-163, integer to τNAF converter	Altera Stratix II EP2S60F1020C4	329	948 ALUTs + 683 Regs
[10]	NIST K-163, τ-adic to integer converter	Altera Stratix II EP2S60F1020C4	481	850 ALUTs + 491 Regs
[1]	NIST K-163, integer to τNAF converter	Xilinx Virtex-4 XC4VLX200	169	1219 slices
[37]	NIST K-233, integer to τNAF converter	Xilinx Virtex-4 XC4VLX200	241	1582 slices

agation; whereas an addition of one to $2d_1$ involves no carry propagation. This difference may be detected using simple power analysis and could potentially leak information about the τ-adic representation.

Computing Alg. 9 requires 121,000 clock cycles and a modular multiplication $a \times b \bmod q$ takes 73,000 clock cycles.

Remark 5 The options of Fig. 1(b) and 1(c) can be combined. We select a random K and execute Lines 1–4 of Alg. 9. Instead of computing Line 5, we use the idea from [31] and compute $s_d = bK = bd_0 + bd_1\tau$. Then, we send bd_0 and bd_1 to the server who computes the integer equivalent. This reduces the latency of the option of Fig. 1(b) but requires more communication.

9 Results and Discussion

We described the circuitries of Fig. 5 and the corresponding ones for $\mu = -1$ in VHDL. We synthesized the code with Synopsys Design Compiler D-2010.03-SP4 and Faraday FSC0L standard cell libraries for UMC 0.13 μm CMOS by using the 'compile ultra' process without additional constraints. We performed simulations with ModelSim SE 6.6d.

Table 3 shows the areas of the datapath extensions for Algs. 1 and 6 for both $\mu = \pm 1$. Because partial τ-adic expansions offer both faster and more secure implementations, we provided the unrolled datapath extension only for Alg. 6. The areas for Alg. 1 were 75.25 and 76.25 gate equivalents (GE) for $\mu = 1$ and $\mu = -1$, respectively. The corresponding areas of Alg. 6 are 128.00 and 114.75 GE so there is a price to pay for the lower latency and resistance against side-channel attacks. However, even these areas are small enough to be embedded into the datapath of a lightweight ALU. For instance, the ALU used in [38] had an area of about 4,323 GE in the same ASIC process and it includes the logic needed for the conversion. A datapath extension with $\omega = 4$ would, hence, represent only about 6 % of this area and the overhead would be even smaller because the converter logic used in [38] could be removed. Tables 1 and 3 show that unrolling provides significant improvements in latency at a relatively low increase in area requirements.

Table 3 Areas of the datapath extensions for Alg. 1 shown in Fig. 5(a) and Alg. 6 shown in Fig. 5(b) on 130 nm CMOS

		$\mu = 1$	$\mu = -1$
Alg. 1	$\omega = 1$	75.25 GE	76.25 GE
Alg. 6	$\omega = 1$	128.00 GE	114.75 GE
	$\omega = 2$	191.25 GE	166.75 GE
	$\omega = 4$	278.25 GE	261.75 GE
	$\omega = 8$	461.25 GE	444.75 GE
	$\omega = 16$	827.75 GE	792.75 GE

9.1 Latencies of ECDSA Signature Generation

To compare with the converters from Sect. 8, we consider computation of $b \times K$ that is required by the ECDSA signature generation. For NIST K-283 and a 16-bit single-port RAM, Algs. 7 and 8 compute it with latencies ranging from 41,568 to 205,597 clock cycles as shown in Table 1. The converters from [38] and Sect. 8.1 compute the conversions in 78,000 and 121,000 clock cycles, respectively. However, a modular multiplication $bk \bmod q$ is also required and it takes about 73,000 clock cycles. This gives total latencies of about 151,000 and 194,000 clock cycles, respectively. Both converters also require datapath extensions similarly to the new algorithms and they cannot be computed as efficiently with standard ALUs. Hence, the proposed techniques are faster and improve upon solutions based on conversions when the unrolling factor $\omega \geq 2$. The above comparison is collected in Table 4.

9.2 Power and Energy

Power and energy consumption are essential characteristics for lightweight implementations. In the case of τ-adic arithmetic, they depend strongly on the ALU, which uses the datapath extensions, as well as on the type of memory, etc. Hence, in order to give exact numbers, we would need to implement an entire ECC ALU and even in that case the numbers would represent only

Table 4 Latency comparison of different options for computing $b \times k$ or $b \times K$ for NIST K-283 with 16-bit ALU and a single-port RAM. Conv. and mult. denote the number of clock cycles for computing the conversions (to either direction) and multiplications with integers or partial τ-adic expansions.

Option	Conv.	Mult.	Total
Fig. 1(a) with [38]	78,000	73,000	151,000
Fig. 1(b) with Sect. 8.1	121,000	73,000	194,000
Fig. 1(c) ($\omega = 1$)	—	206,000	206,000
Fig. 1(c) ($\omega = 2$)	—	126,000	126,000
Fig. 1(c) ($\omega = 4$)	—	85,000	85,000
Fig. 1(c) ($\omega = 8$)	—	66,000	66,000
Fig. 1(c) ($\omega = 16$)	—	55,000	55,000

the design choices taken in designing that specific ALU. This prevents us from giving accurate numbers, but we discuss these issues and provide rough estimates of the effects of τ-adic arithmetic on power and energy consumption in the following.

For this analysis, we assume that power consumption is proportional to the area of the active part of the circuit. Consider, for instance, the ALU of [38] which computes conversions and modular integer multiplications with a 16-bit adder/subtractor but uses a 16-bit binary multiplier for scalar multiplications. Then, the power consumption of the conversion P_c is dominated by the adder/subtractor and the power consumption of the scalar multiplication P_s is determined mostly by the multiplier. The power of τ-adic arithmetic P_τ is dominated by the datapath extension. Hence, we assume $P_c \sim A_c = 138.25\,\mathrm{GE}$[1], $P_m \sim A_m = 856.5\,\mathrm{GE}$[1], and $P_\tau \sim A_\tau = 114.75$–$792.75\,\mathrm{GE}$, where A_c, A_m, and A_τ are the areas of a 16-bit adder/subtractor, a 16-bit binary multiplier, and the datapath extension (see Table 3), respectively. This shows that τ-adic arithmetic uses less power than conversions ($P_\tau < P_c$) only if $\omega = 1$. However, peak power is usually more important than average power for lightweight applications (e.g., passive RFID tags). Hence, the power consumptions of τ-adic arithmetic and conversions are less important in practice because both $P_\tau < P_m$ and $P_c < P_m$ and scalar multiplication determines the peak power consumption.

If a device is battery-powered, then energy consumption is more important than power consumption. Estimates for energy consumptions are obtained by multiplying the above areas with the latencies from Table 4. They show that τ-adic arithmetic reduces energy consumption compared to the option of Fig. 1(b) if $\omega < 8$. The energy consumption of τ-adic arithmetic is on the same level (or only slightly higher) than that of the option of Fig. 1(a) for $\omega < 8$. Therefore, τ-adic arithmetic offers lower latencies without using (significantly) more energy. However, it is clear that scalar multiplication will dominate also energy consumption because it has both significantly longer latency and larger average power consumption.

[1] Obtained by synthesizing 16-bit adder/subtractor and 16-bit binary multiplier codes for 130 nm CMOS using the same setup as above.

10 Conclusions

We provided a comprehensive set of algorithms and hardware architectures for arithmetic with τ-adic expansions. They allow delegating conversions from a constrained device (e.g., an RFID tag or a sensor node) to a more powerful party (e.g., a server). In particular, we showed that, e.g., ECDSA signatures can be computed with low latency and without leaking the secrets through side-channels by using partial τ-adic expansions and unrolled datapath extensions.

We showed that τ-adic arithmetic improves over previous options for implementing Koblitz curve cryptography in lightweight applications. It allows both faster operations and lower power consumption (with different choices for ω) with similar energy consumption levels compared to previous options based on conversions. Hence, τ-adic arithmetic opens up possibilities for trade-offs that are not available by using conversions.

We also showed that Koblitz curves are feasible for lightweight applications even when modular integer arithmetic is required. All that is needed are small datapath extensions for implementing either τ-adic arithmetic or conversions. The use of Koblitz curves and our techniques based on the partial τ-adic expansions can offer major improvements over general elliptic curves in lightweight cryptosystems because Koblitz curves require considerably less computation for similar security levels, which leads to direct improvements, especially, in energy consumption of scalar multiplication.

Acknowledgements This work was done when K. Järvinen was an FWO Pegasus Marie Curie Fellow. S. Sinha Roy was supported by the Erasmus Mundus PhD Scholarship. The work was partly funded by KU Leuven under GOA TENSE (GOA/11/007) and the F+ fellowship (F+/13/039) and by the Hercules Foundation (AKUL/11/19).

We thank one of the anonymous reviewers of a preliminary version of this paper for pointing out the option of Remark 5.

References

1. Adikari, J., Dimitrov, V., Järvinen, K.: A fast hardware architecture for integer to τNAF conversion for Koblitz curves. IEEE Trans. Comput. 61(5), 732–737 (May 2012)

2. Ahmadi, O., Hankerson, D., Rodríguez-Henríquez, F.: Parallel formulations of scalar multiplication on Koblitz curves. J. Univ. Comput. Sci. 14(3), 481–504 (2008)
3. Aranha, D.F., Faz-Hernández, A., López, J., Rodríguez-Henríquez, F.: Faster implementation of scalar multiplication on Koblitz curves. In: Progress in Cryptology (LATINCRYPT 2012), LNCS, vol. 7533, pp. 177–193. Springer (2012)
4. Azarderakhsh, R., Järvinen, K.U., Mozaffari-Kermani, M.: Efficient algorithm and architecture for elliptic curve cryptography for extremely constrained secure applications. IEEE Trans. Circuits Syst. I, Reg. Papers 61(4), 1144–1155 (Apr 2014)
5. Azarderakhsh, R., Reyhani-Masoleh, A.: High-performance implementation of point multiplication on koblitz curves. IEEE Trans. Circuits Syst. II 60(1), 41–45 (2013)
6. Batina, L., Mentens, N., Sakiyama, K., Preneel, B., Verbauwhede, I.: Low-cost elliptic curve cryptography for wireless sensor networks. In: Proc. 3rd Europ. Workhop Security and Privacy in Ad-Hoc and Sensor Networks (ESAS 2006). LNCS, vol. 4357, pp. 6–17 (2006)
7. Bauer, A., Jaulmes, E., Prouff, E., Reinhard, J.R., Wild, J.: Horizontal collision correlation attack on elliptic curves. Cryptography and Communications 7(1), 91–119 (2015)
8. Benits, Jr., W.D., Galbraith, S.D.: The GPS identification scheme using Frobenius expansions. In: West. Europ. Workshop Research in Cryptology (WEWoRC'07). LNCS, vol. 4945, pp. 13–27 (2008)
9. Brumley, B.B., Järvinen, K.: Koblitz curves and integer equivalents of Frobenius expansions. In: Selected Areas in Cryptography (SAC 2007). LNCS, vol. 4876, pp. 126–137 (2007)
10. Brumley, B.B., Järvinen, K.U.: Conversion algorithms and implementations for Koblitz curve cryptography. IEEE Trans. Comput. 59(1), 81–92 (Jan 2010)
11. Cinnati Loi, K.C., An, S., Ko, S.B.: FPGA implementation of low latency scalable elliptic curve cryptosystem processor in $GF(2^m)$. In: IEEE Int. Symp. Circuits and Systems (ISCAS 2014). pp. 822–825. IEEE (2014)
12. Cinnati Loi, K.C., Ko, S.B.: High performance scalable elliptic curve cryptosystem processor for Koblitz curves. Microproc. Microsyst. 37(4), 394–406 (2013)
13. De Clercq, R., Uhsadel, L., Van Herrewege, A., Verbauwhede, I.: Ultra low-power implementation of ECC on the ARM Cortex-M0+. In: Design Automation Conference (DAC 2014). pp. 1–6. ACM (2014)
14. Hankerson, D., Hernandez, J.L., Menezes, A.: Software implementation of elliptic curve cryptography over binary fields. In: Cryptographic Hardware and Embedded Systems (CHES 2000). LNCS, vol. 1965, pp. 1–24. Springer (2000)
15. Hanley, N., Kim, H., Tunstall, M.: Exploiting collisions in addition chain-based exponentiation algorithms using a single trace. In: Topics in Cryptology — CT-RSA 2015. Lecture Notes in Computer Science, vol. 9048, pp. 431–448. Springer (2015)
16. Hanser, C., Wagner, C.: Speeding up the fixed-base comb method for faster scalar multiplication on Koblitz curves. In: Modern Cryptography and Security Engineering (MoCrySEn 2013), LNCS, vol. 8128, pp. 168–179. Springer (2013)
17. Hein, D.M., Wolkerstorfer, J., Felber, N.: ECC is ready for RFID - a proof in silicon. In: Selected Areas in Cryptography (SAC 2008). LNCS, vol. 5381, pp. 401–413 (2009)
18. Järvinen, K.: Optimized FPGA-based elliptic curve cryptography processor for high-speed applications. Integration 44(4), 270–279 (2011)
19. Järvinen, K., Forsten, J., Skyttä, J.: Efficient circuitry for computing τ-adic non-adjacent form. In: Proc. 13th IEEE Int. Conf. Electronics, Circuits and Systems (ICECS 2006). pp. 232–235 (2006)
20. Järvinen, K., Verbauwhede, I.: How to use Koblitz curves on small devices? In: Smart Card Research and Advanced Application Conf. (CARDIS 2014). LNCS, vol. 8968, pp. 154–170 (2015)
21. Joye, M., Tymen, C.: Compact encoding of non-adjacent forms with applications to elliptic curve cryptography. In: Public Key Cryptography (PKC 2001). LNCS, vol. 1992, pp. 353–364 (2001)
22. Koblitz, N.: Elliptic curve cryptosystems. Math. Comput. 48(177), 203–209 (1987)
23. Koblitz, N.: CM-curves with good cryptographic properties. In: CRYPTO '91. LNCS, vol. 576, pp. 279–287 (1991)
24. Koçabas, Ü., Fan, J., Verbauwhede, I.: Implementation of binary Edwards curves for very-constrained devices. In: Proc. 21st IEEE Int. Conf. Application-specific Systems Architectures and Processors (ASAP 2010). pp. 185–191 (2010)
25. Lange, T.: Koblitz curve cryptosystems. Finite Fields Th. App. 11, 200–229 (2005)
26. Lee, Y.K., Sakiyama, K., Batina, L., Verbauwhede, I.: Elliptic-curve-based security processor for RFID. IEEE Trans. Comput. 57(11), 1514–1527 (2008)
27. Lutz, J., Hasan, A.: High performance FPGA based elliptic curve cryptographic co-processor. In: Int. Conf. Information Technology: Coding and Computing (ITCC 2004). vol. 2, pp. 486–492. IEEE (2004)
28. Meier, W., Staffelbach, O.: Efficient multiplication on certain nonsupersingular elliptic curves. In: CRYPTO '92. LNCS, vol. 740, pp. 333–344 (1993)
29. Miller, V.S.: Use of elliptic curves in cryptography. In: CRYPTO '85. LNCS, vol. 218, pp. 417–426 (1986)
30. Montgomery, P.L.: Speeding the Pollard and elliptic curve methods of factorization. Math. Comput. 48, 243–264 (1987)
31. Naccache, D., M'Raïhi, D., Vaudenay, S., Raphaeli, D.: Can D.S.A. be improved? Complexity trade-offs with the digital signature algorithm. In: EUROCRYPT '94. LNCS, vol. 950, pp. 77–85 (1994)
32. National Institute of Standards and Technology (NIST): Digital signature standard (DSS). FIPS PUB 186-4 (Jul 2013)
33. Okada, S., Torii, N., Itoh, K., Takenaka, M.: Implementation of elliptic curve cryptographic coprocessor over $GF(2^m)$ on an FPGA. In: Cryptographic Hardware and Embedded Systems (CHES 2000), LNCS, vol. 1965, pp. 25–40. Springer (2000)
34. Okeya, K., Takagi, T., Vuillaume, C.: Efficient representations on Koblitz curves with resistance to side channel attacks. In: Proc. 10th Australasian Conf. Information Security and Privacy (ACISP 2005). LNCS, vol. 3574, pp. 218–229 (2005)
35. Oren, Y., Feldhofer, M.: A low-resource public-key identification scheme for RFID tags and sensor nodes. In: ACM Conf. Wireless Network Security (WiSec'09). pp. 59–68. ACM (2009)
36. Secunet Security Networks AG: Elliptic curve cryptography "Made in Germany". Press release (2014), online: `https://www.secunet.com/fileadmin/user_upload/Presse/Pressemitteilungen/Pressemitteilungen_`

EN/Pressemitteilungen_2014_EN/140625_PI_ECC_EN.pdf, retrieved Feb. 21, 2017
37. Sinha Roy, S., Fan, J., Verbauwhede, I.: Accelerating scalar conversion for Koblitz curve cryptoprocessors on hardware platforms. IEEE Trans. VLSI Syst. 23(5), 810–818 (2015)
38. Sinha Roy, S., Järvinen, K., Verbauwhede, I.: Lightweight coprocessor for Koblitz curves: 283-bit ECC including scalar conversion with only 4.3 kGE. In: Cryptographic Hardware and Embedded Systems (CHES 2015). LNCS, vol. 9293, pp. 102–122 (2015)
39. Solinas, J.A.: Efficient arithmetic on Koblitz curves. Design Code Cryptogr. 19(2–3), 195–249 (2000)
40. Taverne, J., Faz-Hernández, A., Aranha, D.F., Rodríguez-Henríquez, F., Hankerson, D., López, J.: Speeding scalar multiplication over binary elliptic curves using the new carry-less multiplication instruction. J. Cryptogr. Eng. 1(3), 187–199 (2011)
41. Vuillaume, C., Okeya, K., Takagi, T.: Defeating simple power analysis on Koblitz curves. IEICE Trans. Fund. Elect. E89-A(5), 1362–1369 (May 2006)
42. Weimerskirch, A., Stebila, D., Shantz, S.C.: Generic $GF(2^m)$ arithmetic in software and its application to ECC. In: Australasian Conference on Information Security and Privacy (ACISP 2003). LNCS, vol. 2727, pp. 79–92. Springer (2003)